

lutional encoder (used in transmitter path) and a Viterbi decoder (used in receive path). Interleaving may be done for the data, which helps in spreading the error over time, thereby helping the receiver de-interleave and decode the frame correctly.

[0041] RF circuitry block **340** includes an RF up-converter and an RF down-converter. For a GSM system, the RF up-converter converts modulated baseband signals (I and Q) either at zero intermediate frequency (IF) or some IF to RF frequency (890-915 MHz). The RF down-converter converts RF signals (935 to 960 MHz) to baseband signals (I and Q). For a GSM system, GMSK modulation is used.

[0042] Antenna **395** is a metallic object that converts and electro-magnetic signal to and electric signal and vice versa. Commonly used antennas may include a helix type, a planar inverted F-type, a whip, or a patch type. Microstrip patch type antennas are popular among mobile phones due to small size, easy integration on a printed circuit board and multi-frequency band of operation. In a preferred embodiment of mobile phone **121**, antenna **395** may support different wire-area standards, including GSM, CDMA, LTE, and WiMAX, as well as short-range standards, including WiFi (WLAN), Bluetooth, and so on.

[0043] If antenna **395** comprises only one antenna used for both transmit and receive operations at different times, the TX/RX switch **345** couples both the transmit (TX) path and the receive (RX) path to antenna **395** at different times. TX/RX switch **345** is controlled automatically by DSP **320** based on a GSM frame structure with respect to the physical slot allocated for that particular GSM mobile phone in both the downlink and the uplink. For frequency division duplexing (FDD) systems, TX/RX switch **345** may be implemented as a diplexer that acts as filter to separate various frequency bands.

[0044] FIG. 4 illustrates the operation of Android™ package manager **400** in exemplary mobile phone **121** according to embodiments of the disclosure. In mobile phone **121**, Android™ package manager **400** manages all applications. Android™ package manager **400** includes package optimizer function **420** that receives original Android™ APK application package **410**, which has a .apk extension and generates therefrom an optimized Android™ APK application package **430**, which has the same name and a .apk extension. The applications handled by Android™ package manager **400** include preloaded system applications and update applications from Android Market™. Packages.xml file **450** is an XML file that contains information about installed packages and is managed by Android™ package manager **400**. Package manager install function **400** installs the optimized APK application package **440** and stores information about installed packages in packages.xml file **450**.

[0045] Android™ package manager **400** parses and edits preloaded system applications and update applications in a way that reduces file size by removing unnecessary resource without affecting the functionality of the application in the target device. Android™ package manager **400** eliminates duplicate and unused resources, while still enabling applications to be updated through, for example, Google Play™ Store. After unnecessary resources are identified and removed, Android™ package manager **400** re-packages the modified and optimized APK application package with a different signature or certificate because the original signing key (private key) is not available. However, Android™

package manager **400** ensures that the optimized APK package with a different signature will still update through Google Play™ Store. Special key/retrieved key table **460** stores a mapping between the original certificates/keys and the newly generated certificates/keys generated from the optimized APK application package.

[0046] FIG. 5 is a flow diagram illustrating the operation of package optimizer function **420** in reducing APK package resources and size according to the principles of the present disclosure. Initially, Android™ package manager **400** adds a package to the queue for the installation process and determine the appropriate location, if needed, and determines the type of installation (i.e., install or update). Android™ package manager **400** may query package-related information to from packages.xml file **450**, which may have an original package.

[0047] To optimize an original APK application package, Android™ package manager **400** executes package optimizer function **420**. Package optimizer function **420** initially unzips a new or a preloaded .apk package (step **510**). Next, package optimizer function **420** parses certificate-related information from the META-INF folder and then deletes the META-INF folder (step **520**). Package optimizer function **420** then scans the RES (resource) folder and analyzes the resource files and elements based on selected target device parameters (e.g., DPI, MCC, MNC, etc.) (step **530**). Package optimizer function **420** deletes unnecessary resources from the RES folder based on a pre-determined policy configuration (step **540**). The policy governs which resources should remain, considering the fact that only certain resources will be used on the target mobile device **121**. By way of example, package optimizer function **420** handles graphic assets efficiently because the graphic assets are related to device DPI and the DPI information does not change. Carrier information, such as MCC and MNC, and some layout .xml data with device DPI also may be considered as unchanged values. Therefore, package optimizer function **420** may safely remove these resource if, for example, device DPI or carrier information is known.

[0048] After the unnecessary resources files and other elements have been removed, package optimizer function **420** zips all of the remaining data into an optimized APK application package having the same name as the original APK application package **410** and the .apk extension (step **550**). Package optimizer function **420** then signs the optimized APK application package **430** with a special key (step **560**). The special key is intended to distinguish from other regular applications. With the special key, mobile phone **121** can determine if the APK application package uses the Reduce Storage Usage tool or not. Package manager install function **400** may then install the optimized APK application package **430** and reflect the installation information on packages.xml file **450** along with certificate-related information from the META-INF retrieved from the original APK package **410**. Package optimizer function **420** creates a mapping table in special key/retrieved key table **460** between the original certificates and the newly created certificates (step **570**). The mapping information is used to get updates from Android Market™, since Android Market™ only has the original certificate information.

[0049] Although the present disclosure has been described with an exemplary embodiment, various changes and modifications may be suggested to one skilled in the art. It is